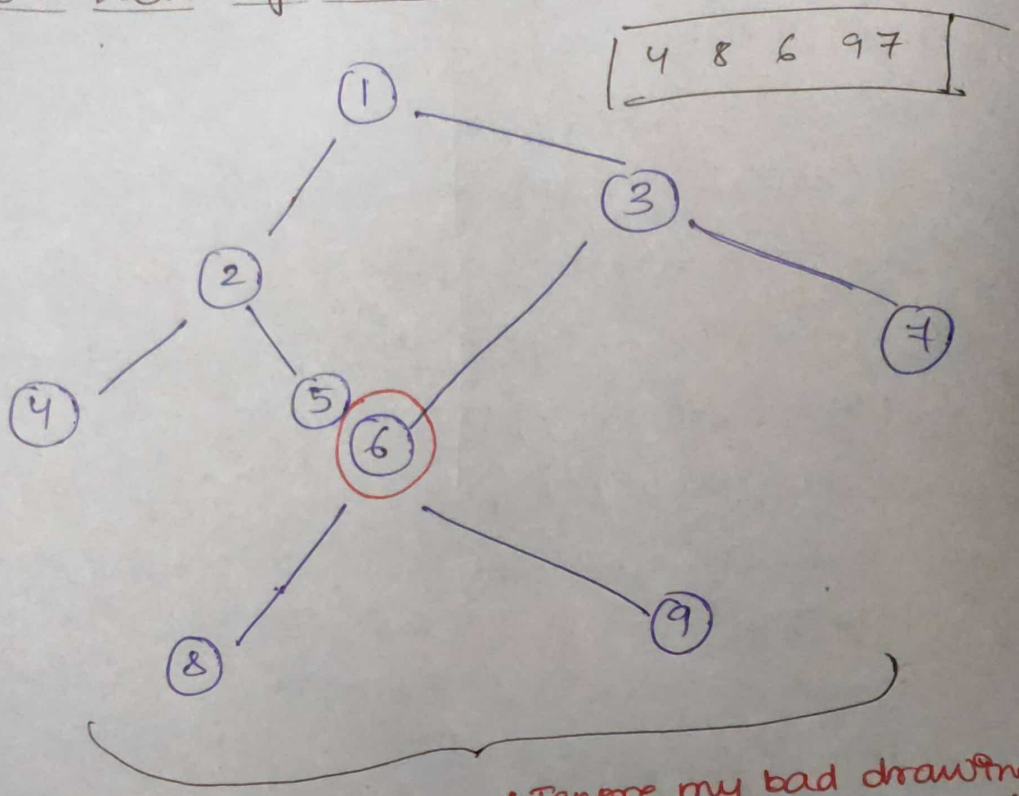
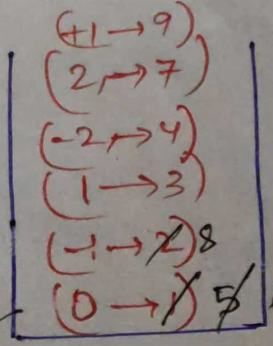
Bottom view of Binary Tree



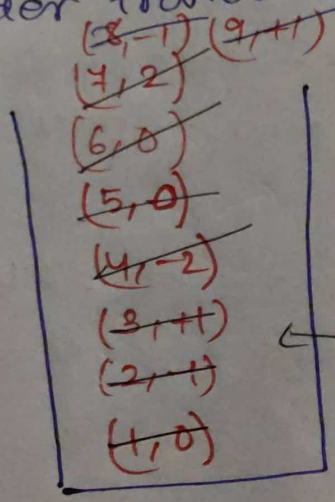
*Ignore my bad drawing

(view from bottom)

Same logic of vertical order traversal.



(line, node)
map (store sorted keys)



Queue.

push into queue.

node = 1 2 3 4 5 6 7

check left & right

replace 0→5 in map

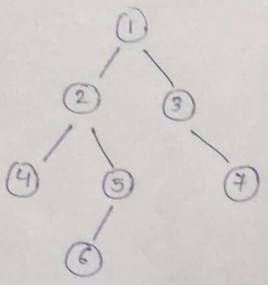
4	8	6	9	7

sorted acc to keys

Code:-

```
vector <int> bottomView (Node *root) {
    vector <int> ans;
    if (root == NULL) return ans;
  map<int,int> mp;
  queue <pair, Node * int> q;
    q.push ({root, 0});
  while (!q.empty()){
        auto it = q.front();
        q.pop();
        Node *node = it.first;
        int line = it.second;
        mp[line] = node->data;
    if (node->left != NULL){
        q.push ({node->left, line-1});
    }
    if (node->right != NULL) {
          q.push ({node->right, line+1});
        }
  }
  for (auto it : mp) {
        ans.push_back (it.second);
      }
  return ans;
}
```

Right-Left Side View:-



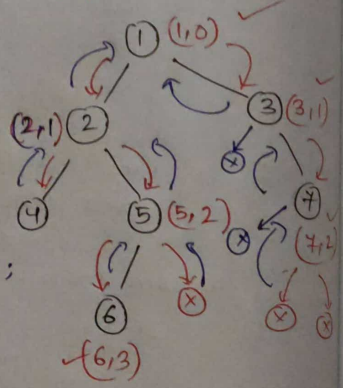
Right view: 1 3 7 6
Left view: 1 2 4 6

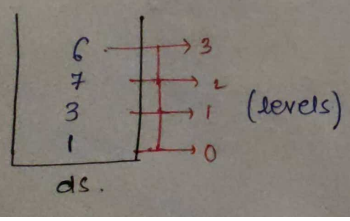
We can use the recursive / iterative technique

- recursive → pre-order (in reverse) → Root left Right. Instead of this we will do Root Right left. TC: O(n)
- iterative → level order.

Using recursive method for crisp & short code.

```
f(node, level) {
  if(node == null)
      return;
  if (level == ds.size)
     ds.add(node)
  f(node->right, level+1);
  f(node->left, level+1);
}   // Pseudo Code.
```





For left view, just call left first.

Code:-

```
vector<int> rightSideView(TreeNode *root) {
    vector<int> res;
    recursion(root, 0, res);
    return res;
}

void recursion(TreeNode *root, int level,
               vector<int>& res) {
    if(root == NULL) return;
    if(res.size() == level) res.push_back(root->val);
    recursion(root->right, level+1, res);
    recursion(root->left, level+1, res);
}
```